Proportional Rate Reduction for TCP

draft-ietf-tcpm-proportional-rate-reduction-01

Matt Mathis, Nandita Dukkipati, Yuchung Cheng
{mattmathis, nanditad, ycheng}@google.com

TCPM, IETF-83
Mar 30, 2011

Motivation

- Two widely deployed algorithms for loss recovery: RFC 3517 fast recovery and Rate Halving.
- Both are prone to timeouts in some situations.
  - RFC 3517 fast recovery waits for half of the ACKs to pass before sending data.
  - Rate halving does not compensate for implicit cwnd reduction under heavy losses.
- Goals of Proportional Rate Reduction.
  - Reduce timeouts by avoiding excessive window reductions.
  - Converge to cwnd chosen by congestion control.

All positive results

- Better than Rate-Halving
  - More accurate cwnd reduction
  - Less prone to timeouts
  - Avoids excess recovery time due to depressed cwnd
  - Better overall performance
- Better than RFC3517
  - Gradual cwnd reduction (no silence)
  - Less prone to timeout following burst losses
  - Does not cause bursts when losses > CC reduction
  - Less prone to lost retransmissions
  - Better overall performance

Ready for WGLC(?)

- So far the only comments have been nits
- and discussion of some previously rejected alternative designs

Backup slides (from IETF-81)

Proportional Rate Reduction

- Two separate phases:
- pipe > ssthresh
  - **Proportional rate reduction** (PRR) algorithm:
  - Patterned after rate halving but at rate chosen by CC
  - Main idea: sending_rate = (CC_reduction_factor) * (data_rate_at_the_receiver)
  - Faster but smother than 3517
- pipe < ssthresh
  - **SlowStart Reduction Bound** (SSRB)
    - Open window by one segment per ACK
  - Main purpose: bring pipe back up to ssthresh
  - Less aggressive than 3517

Slowstart while in Recovery?

- RFC3517 sends bursts! Consider:
  - It sends (ssthresh-pipe) segments
    - On every ACK, when positive
  - Start with pipe=cwnd=100 segments
  - Loose segments 1-90
  - ACK from segment 93 can trigger 40 segments
  - Not conservative

- Proposed behavior (SSRB)
  - Pure packet conservation, plus one extra segment
    - Send the same quantity of data as ACK'd +1
    - For the next ~40 ACKs

Algorithm comparisons - Single loss

At beginning: cwnd = FlightSize = pipe = 20

```
RFC 3517
ack#   X  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
cwnd:    20 20 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
pipe:    19 19 18 18 17 16 15 14 13 12 11 10 10 10 10 10 10 10 10
sent:     N  N  R                    N  N  N  N  N  N  N  N

Rate halving (Linux)
ack#   X  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
cwnd:    20 20 19 18 18 17 17 16 16 15 15 14 14 13 13 12 12 11 11
pipe:    19 19 18 18 17 17 16 16 15 15 14 14 13 13 12 12 11 11 10
sent:     N  N  R    N    N    N    N    N    N    N    N

PRR
ack#   X  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
pipe:    19 19 18 18 18 17 17 16 16 15 15 14 14 13 13 12 12 11 10
sent:     N  N  R    N    N    N    N    N    N       N  N
RB:                                          s  s
```

Algorithm comparisons - 15 Losses

At beginning: cwnd = FlightSize = pipe = 20

RFC 3517

ack#	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	15	16	17	18	19
cwnd:										20	20	11	11	11						
pipe:										19	19	4	10	10						
sent:										N	N	7R	R	R						

Rate Halving (Linux)

ack#	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	15	16	17	18	19
cwnd:										20	20	5	5	5						
pipe:										19	19	4	4	4						
sent:										N	N	R	R	R						

PRR-SSRB

ack#	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	15	16	17	18	19
pipe:										19	19	4	5	6						
sent:										N	N	2R	2R	2R						

Is the high loss case commom? YES

- Statistics of pipe vs ssthresh at FR
  - pipe > ssthresh: 45%
  - pipe = ssthresh: 13%
  - pipe < ssthresh: 32%
  - (also have distribution data on pipe-ssthresh)
- Losses exceed net CC window reduction for 32% of FR
  - RFC 3517 would send a burst
  - Contributing causes:
    - CUBIC (only 30% window reduction)
    - Better SACK retransmit algorithms
  - Still unexpectedly large

Measurement setup

- Implemented in Linux 2.6
- Experiment on Google Web servers:
    - CUBIC, limited transmit, lost retrans detect
    - Compare:
        - Rate halving (Linux default), RFC 3517, PRR-SSRB
    - Mixed services, excluding streaming video
- (Post experiment) Currently under test to go Google wide

Measurement Summary

- PRR-SSRB does best
- RFC 3517 experiences:
  - 2.6% more timeouts
  - 3% more retransmissions
  - 29% increase detected lost retransmissions
  - Similar transaction (short flow completion) times
    - Being more aggressive has zero net gain
- Rate halving (Linux default) experiences:
  - 5% more timeouts
  - Lower cwnd values at the end of recovery
  - 3-5% longer transaction (short flow completion) times

Details in an upcoming IMC-2011 paper

Conclusion

- PRR-SSRB provides more accurate and smoother window adjustments during loss recovery
- Benefits:
  - Fewer symptoms of bursts (timeouts)
  - Lower tail latencies of request-response traffic.
  - Better performance than rate halving
- Cost:
  - Possible indirect costs to sending packets early during recovery as opposed to letting more ACKs pass first.
- Recommendation
  - Wide testing with PRR-SSRB

Going forward

- Adopt PRR as a WG item(?)
  - Tentative Goal: experimental status.
    - So people can gain experience.

More Details

- Only as time and interest permits
- Outline
    - Packet conservation review
    - PRR pseudo code
    - Other reduction bound varients
    - Properties of all PRR variants
    - Measurement details
    - Example Time Sequence Graphs

Packet conservation review

- Original Van Jacobson concept behind TCP self clock
- Quantity of data sent exactly the same as data reported arriving at the receiver
- A standing queue at a bottleneck will have constant length
- Any more aggressive algorithm will cause queue growth
  - And the potential for "forced losses" in drop tail
- Key concept: DeliveredData computed for each ACK
  - For SACK DeliveredData is not an estimator
    - DeliveredData = delta(snd.una)+delta(SACKd)
    - Can be observed anywhere on the ACK path.
  - Can be estimated for non-SACK
    - Sum over time must match forward progress

PRR pseudo code move

Start of recovery:
ssthresh = CongCtrlAlg() // Target cwnd after recovery.
RecoverFS = snd.nxt - snd.una // FlightSize.
prr_delivered = prr_out = 0 // accounting

On each ACK in recovery, compute:
// DeliveredData: #pkts newly delivered to receiver.
DeliveredData = delta(snd.una) + delta(SACKd)
// Total pkts delivered in recovery.
prr_delivered += DeliveredData
pipe = RFC 3517 pipe algorithm

Algorithm:
```
if (pipe > ssthresh) // Proportional Rate Reduction.
  sndcnt = CEIL(prr_delivered * ssthresh / RecoverFS) - prr_out
else                 // Reduction Bound.
  limit = (Reduction bound algorithm)
  sndcnt = MIN(ssthresh - pipe, limit)
```
On any data transmission or retransmission:
prr_out += (data sent) // Smaller than or equal to sndcnt.

Reduction Bound Variants

- PRR-UB: Unlimited Bound
  - No limit (infinite)
  - Behavior parallels RFC 3517
    - Transmission bursts if pipe falls below ssthresh
  - Included only for reference & comparisons
- PRR-CRB: Conservative Reduction Bound
  - Strong packet conservation properties
    - limit = prr_delivered - prr_out
    - Thus prr_out can never exceed prr_delivered
- PRR-SSRB: Slowstart reduction bound
  - "Grows" the window when pipe < ssthresh
    - Relax CRB by exactly one segment per ACK

Proportional Rate Reduction with unlimited bound (PRR-UB)

- Observation: in some cases RFC 3517 sends bursts
  - Example: pipe == cwnd == 100 packets, lost packets 1-90, packet 93 can generate a burst up to 40 packets.
  - RFC 3517 is not at all conservative in this scenario.
  - PRR-RB bounds #pkts sent by (prr_delivered - prr_out).

- PRR-UB mirrors RFC 3517:
  - Allow arbitrary bursts to bring pipe up to ssthresh

```
if (pipe > ssthresh) // Proportional Rate Reduction
  sndcnt = CEIL(prr_delivered * ssthresh / RecoverFS)- prr_out
else
  sndcnt = ssthresh - pipe
```

Proportional Rate Reduction with Conservative Bound

- Send up to as much data as was (previously) delivered
  - limit = prr_delivered - prr_out
  - prr_out can not become larger than prr_delivered
- Cool properties:
  - Bound is strict packet conserving
  - Constant sized (standing) queue at bottleneck
  - Maximally aggressive w/ causing additional losses
  - Philosophically clean and ideal
- Downside
  - Lower measured perfomance than RFC 3517
    - Burst losses that depress cwnd seem to be common

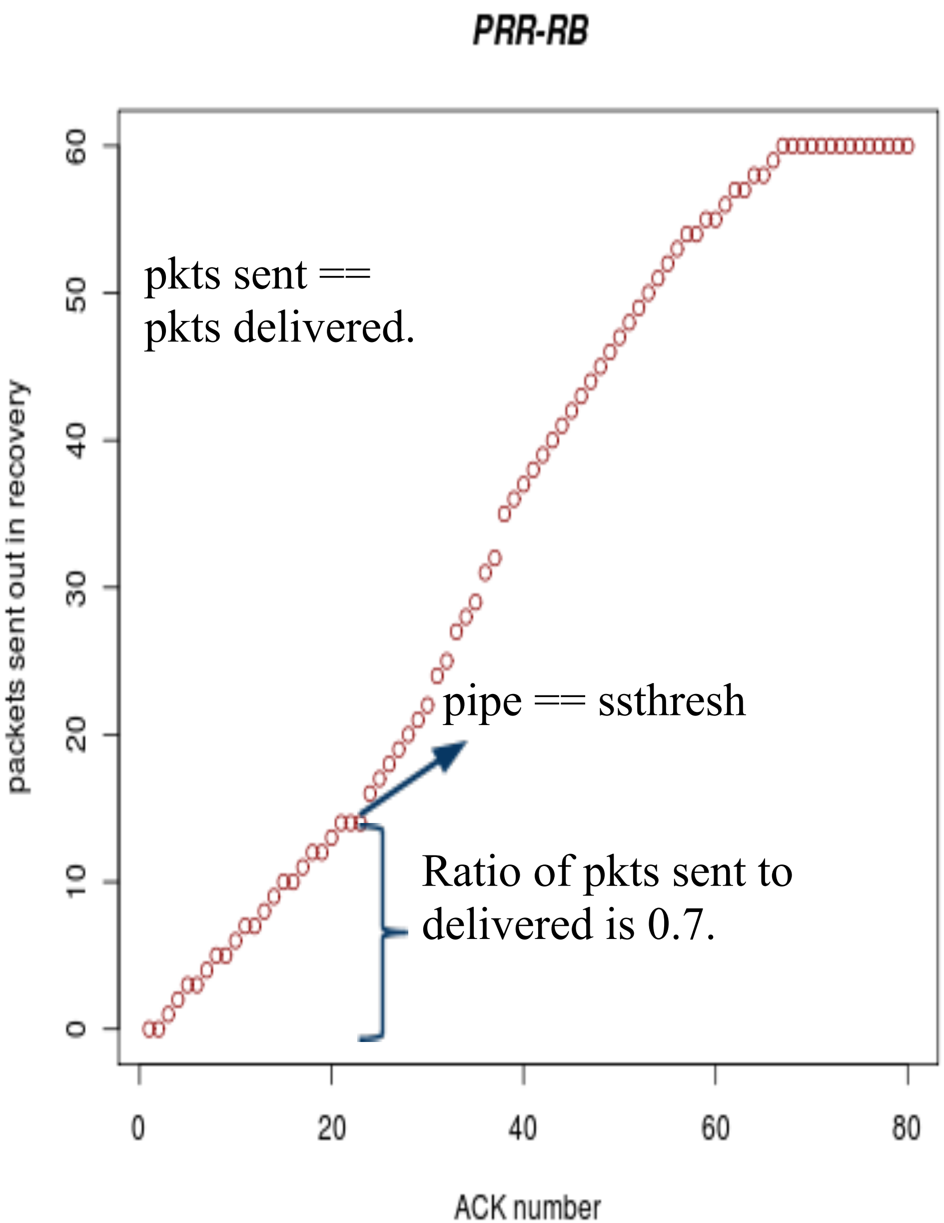
Proportional Rate Reduction with slowstart Bound

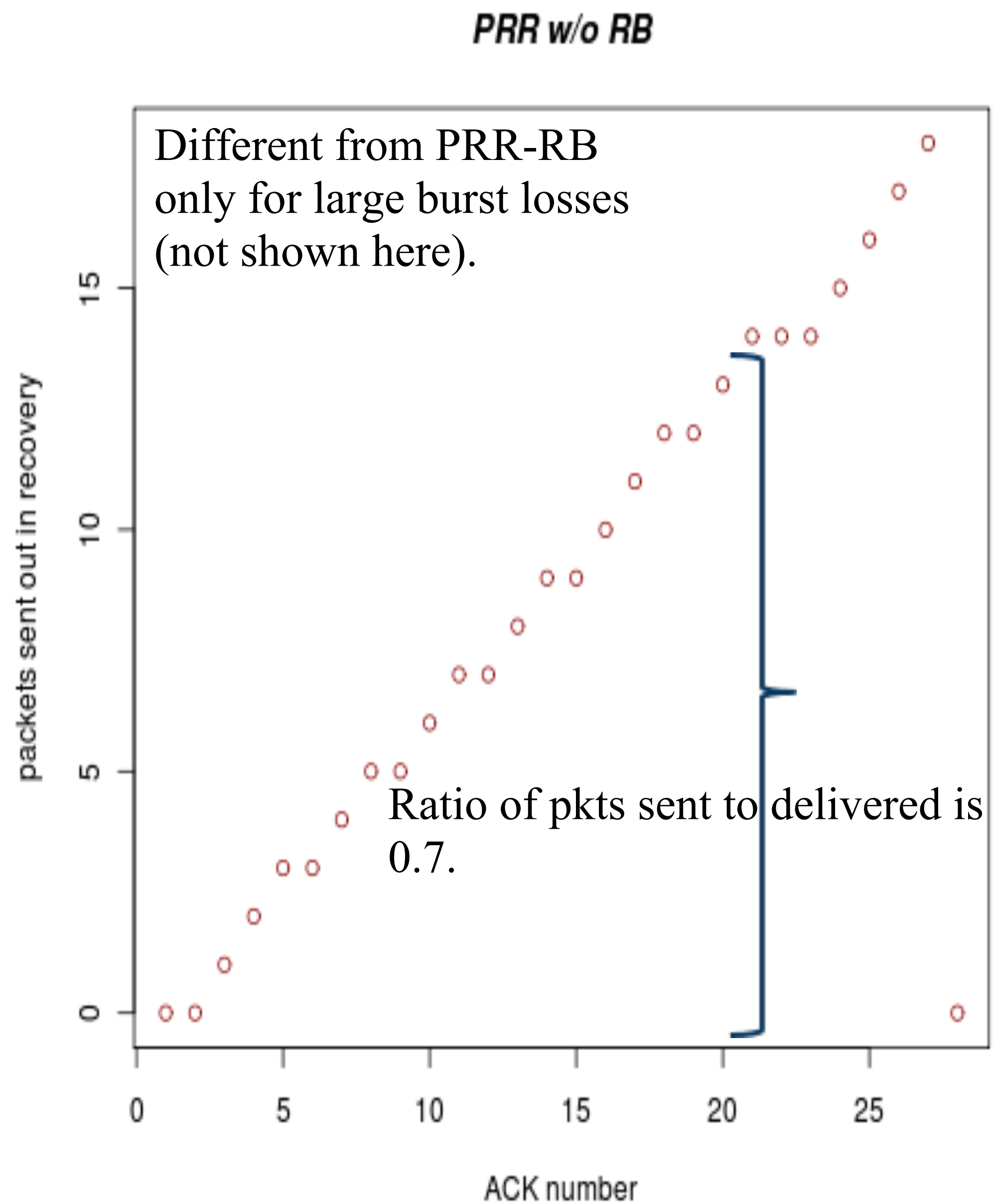
- Relax PRR-CRB by allowing one extra segment per ACK
    - Effectively introduce slowstart after excess loss
    - But no burst as permitted by RFC 3517
- Allowing 1 extra segment is good compromise between:
    - Unlimited extra segments (PRR-UB/RFC 3517)
    - Zero extra segments (PRR-CRB)

Properties of PRR (all variants)

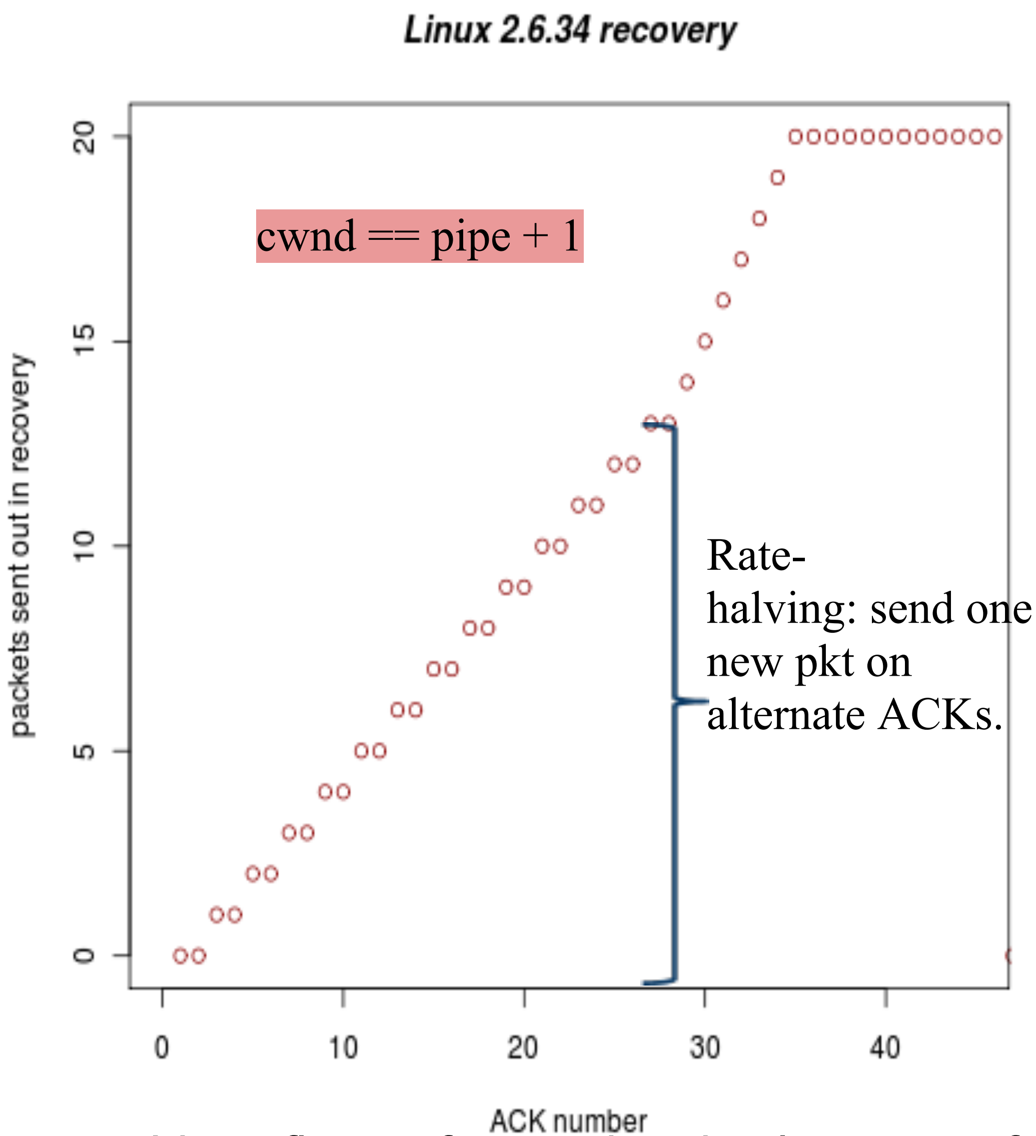
- Spreads out window reduction evenly across the recovery period.
- For moderate loss, converges to target cwnd chosen by CC.
- Maintains ACK clocking even for large burst losses.
- Precision of PRR-RB is derived from DeliveredData
  - Which is not an estimator
- Banks the missed opportunities to send if application stalls during recovery.
- Less sensitive to errors of the pipe estimator.

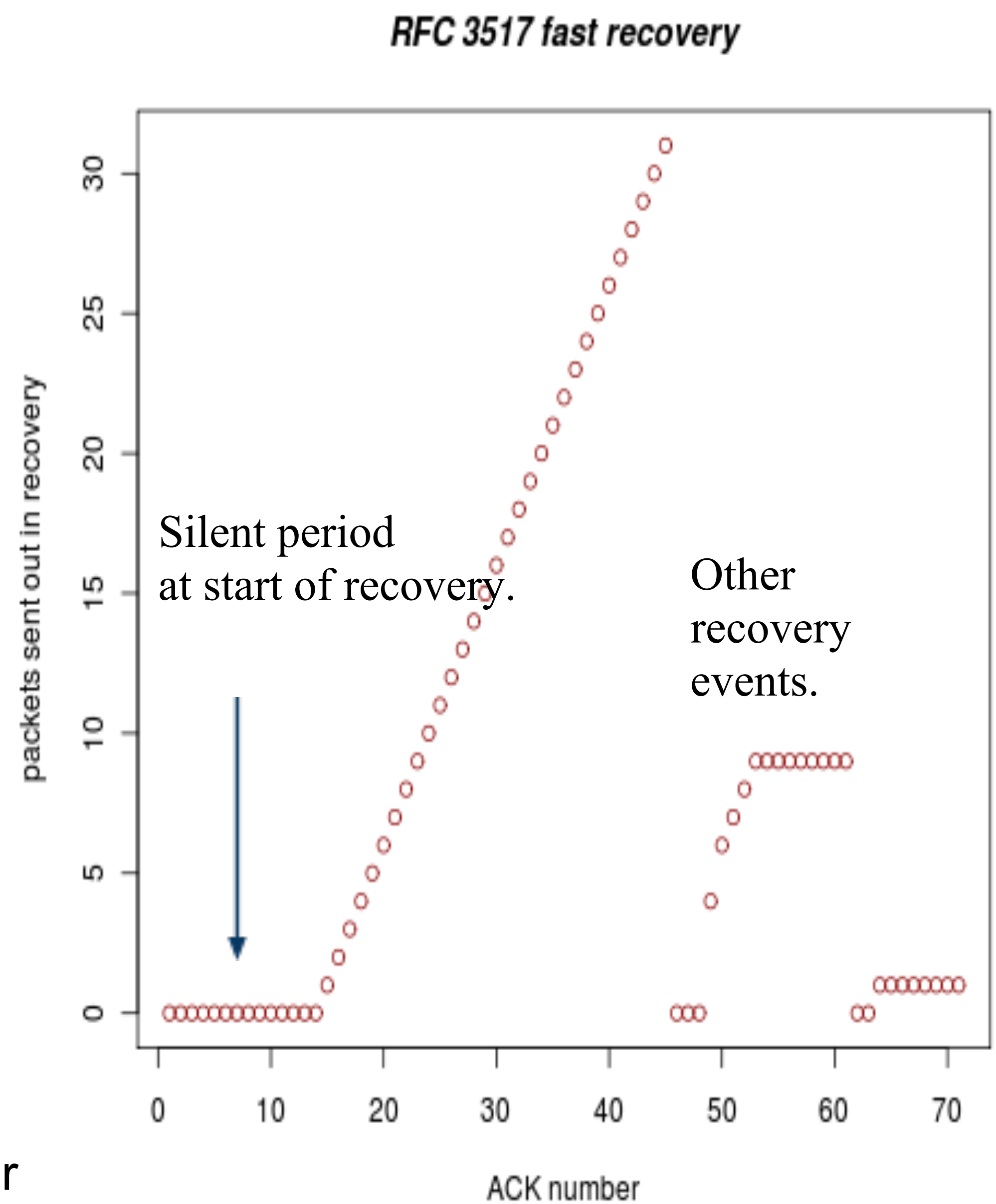
Example flow in PRR-RB, PRR w/o RB





Example flow in Linux and RFC 3517





Linux flows often end up in slow start after recovery when short responses have no new data to send and pipe reduces to 0 .